НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки
6.050101 “Комп’ютерні науки”

на тему: Адаптація акустичної моделі до особливостей звукового сигналу

Виконав: студент 4 курсу, групи ТР-52

Сехін Олексій Петрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Стативка Юрій Іванович _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту
Сехіну Олексію Петровичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Адаптація акустичної моделі до особливостей звукового сигналу”
керівник роботи доцент, к.т.н. Стативка Юрій Іванович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від ___ _____ 201__ р. № _____
2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи адаптована акустична модель
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати проблематику розпізнавання мови в існуючих системах, провести огляд сучасних програмних рішень по адаптації акустичної моделі, реалізувати програмний додаток для адаптації акустичної моделі
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов’язкових креслень)
1. Постановка задачі адаптації акустичної моделі до особливостей звукового сигналу. 2. Огляд проблеми точності розпізнавання мови в тексті. 3. Огляд технологій для вирішення задачі розпізнавання мови та адаптації акустичної моделі. 4. Опис програмної реалізації 5. Висновки.

6. Публікації: _____

Дата видачі завдання ””___ _____ 201__р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2.	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Захист програмного продукту		
7.	Оформлення пояснювальної записки		
8.	Передзахист		
9.	Захист		

Студент _____ Сехін О. П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стативка Ю.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метою роботи була розробка та реалізація програми для адаптації акустичної моделі, що дасть змогу користувачу створювати необхідний набір команд для адаптації, створити всі необхідні дані для адаптації, включаючи можливість запису аудіо файлу для кожної команди та провести адаптацію обраної акустичної моделі. Після проведення адаптації користувач отримує можливість перевірити якість адаптації та точність розпізнавання мови до адаптації, для порівняння. Також программа надасть показник WER, що характеризує кількість неправильно розпізнаних слів.

Ключові слова: розпізнавання мови, WER, адаптація, акустична модель, мовна модель, словник.

Записка містить 64 сторінкок, 13 рисунків, 11 формул та 20 посилань

ABSTRACT

The aim of the work was to develop and implement a program for adapting the acoustic model, which will allow the user to create the necessary set of commands for adaptation, create all the necessary data for adaptation, including the ability to record an audio file for each team and adapt the selected acoustic model. After the adaptation, the user gets an opportunity to check the quality of adaptation and the accuracy of speech recognition before adaptation, for comparison. Also, the program will provide an indicator WER that characterizes the number of incorrectly recognized words.

Keywords: speech recognition, WER, adaptation, acoustic model, language model, vocabulary.

The note contains 64 pages, 13 figures, 11 formulas and 20 reference

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HMM – hidden Markov model, прихована Марківська модель.

MLLR (maximum Likelihood Linear Regression) – метод адаптації, який підходить коли набір даних обмежений.

MAP – maximum A Posteriori. – метод адаптації в якому змінюється кожний параметр акустичної моделі.

ЕОМ – електронно обчислювальні машини.

MFCC (Mel Frequency Cepstral Coefficients) – це деяке уявлення енергії спектра сигналу.

ДПФ – дискретне перетворення Фур’є.

API (application programming interface) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

ВСТУП

Голосовий спосіб взаємодії з технічними та інформаційними системами відкриває багато можливостей для підвищення зручності керування. Технологію голосового вводу застосовують в широкому спектрі систем - від додатків на смартфонах до систем "Розумний будинок". Проте, точність розпізнавання мови сучасними системами в умовах шуму все ще не дозволяє використовувати цю технологію, як основний спосіб передачі інформації комп'ютеру [1]. Тому проблема підвищення ефективності систем розпізнавання мови в умовах засміченого сторонніми шумами сигналу є актуальною по сей день.

Всі існуючі системи розпізнавання мови можна поділити на дві категорії: з відкритим вихідним кодом і системи з закритим вихідним кодом [2]. Реалізація систем розпізнавання мови з закритим вихідним кодом виконана більш якісно, такі системи мають високу точність розпізнавання мови. Найвідоміші з них це: Dragon Mobile SDK, Google Speech Recognition API, Yandex Speech Kit, Microsoft Speech API. Серед акустичних моделей з відкритим вихідним кодом можна виділити: CMUSphinx [3], HTK [4], Kaldi [5], Julius [6]. Такі системи є більш доступні для використання, інтегрування нових можливостей та проведення досліджень.

Для даної роботи було обрано систему CMUSphinx. Цей вибір обумовлений наявністю великої кількості наукового матеріалу і документацій, легкістю інтегрування та зручністю використання.

Для підвищення точності розпізнавання мови в умовах шуму необхідно проводити коригування параметрів акустичної моделі. Для цього існують декілька методів адаптації акустичної моделі до особливостей вхідного сигналу. Найвідоміші з них [7] – MLLR (Maximum Likelihood Linear Regression) – трансформація і MAP (Maximum A Posteriori) адаптація. В даній роботі було використано перший метод. Він полягає в обчисленні матриць лінійних перетворень для кожного стану фонем в

адаптаційних даних.

Метою роботи є створення програми, що дає змогу користувачу створити всі необхідні вхідні данні для адаптування існуючої акустичної моделі, провести адаптацію та порівняти результат роботи вхідної акустичної моделі з адаптованою.

У роботі описується процес перетворення мови в текст та повний цикл проведення адаптації, починаючи від огляду та вибору технологій та програмних засобів, налаштування необхідного програмних модулів та завершуючи описом використання системи майбутнім користувачем.

Зміст розділів даної роботи наступний:

У першому розділі описується постановка задачі реалізації системи, її мета, об’єкт і предмет дослідження, а також завдання дослідження.

У другому розділі приведений огляд алгоритму розпізнавання мови та проблем систем розпізнавання мови.

У третьому розділі проводиться огляд технологій програмування для реалізації системи розпізнавання та адаптації акустичних моделей.

У четвертому розділі описується програмна реалізація додатка, що дозволяє створити адаптацію акустичної моделі, та проведене дослідження якості адаптації в різних умовах шуму.

1. ПОСТАНОВКА ЗАДАЧІ АДАПТАЦІЇ АКУСТИЧНОЇ МОДЕЛІ ДО ОСОБЛИВЕСТЕЙ ЗВУКОВОГО СИГНАЛУ

Використовуючи теоретичний матеріал та існуючі програмні засоби, створити систему для адаптації акустичних моделей, яка буде надавати можливість створення всіх необхідних для адаптації даних, та використовуючи їх, створювати нову, вже адаптовану акустичну модель під особливості даного диктора, шумів та інших факторів навколишнього середовища. Також система повинна надавати можливість порівняти первісну і адаптовану системи та видавати результат покращення рівня розпізнавання.

В результаті розробки програмного продукту, користувачу буде надана можливість створити адаптацію існуючої акустичної моделі під свій голос та під умови навколишнього середовища.

Метою розробки системи адаптування акустичної моделі до особливостей звукового сигналу є створення програмного продукту, що дасть змогу отримати адаптовані акустичні дані, які зможуть покращити роботу алгоритмів по розпізнаванню мови, і тим самим покращити якість розпізнавання.

Об’єктом дослідження є технології розпізнавання мови та адаптації.

Предметом дослідження є система розпізнавання мови CMU Sphinx.

Для досягнення поставленої задачі були сформульовані наступні завдання дослідження, що визначили логіку дослідження та його структуру:

— проаналізувати сучасні технології розпізнавання мови та засоби адаптації, обрати найкращі;

— провести порівняльний аналіз якості розпізнавання мови різних типів існуючих акустичних моделей для російської мови, та обрати найкращу;

— створити систему для адаптації акустичної моделі під особливості звукового сигналу.

Вхідною інформацією вважаються наступні дані:

— набір команд для адаптації;

— запис кожної команди в окремому аудіо файлі.

Вихідною інформацію вважаються наступні дані:

— словник зі списком всіх використовуваних слів поділених на фонеми;

— адаптована мовна модель;

— адаптована акустична модель;

— показник якості адаптації.

Програма повинна мати зрозумілий та зручний інтерфейс для користувача будь-якого віку.

Програма повинна реалізовувати такі функції:

— видавати список речень, необхідних для адаптації системі під вказану предметну область.;

— надавати можливість записати кожне речення в окремий аудіо файл;

— проводити розпізнавання мови вказаного аудіо файлу та в реальному режимі часу, використовуючи адаптовану та не адаптовану моделі;

— переводити розпізнаний текст в спеціальне текстове поле для порівняння;

— надавати результат адаптації;

— надавати зрозумілі повідомлення про помилки.

2. ОГЛЯД ПРОБЛЕМИ ТОЧНОСТІ РОЗПІЗНАВАННЯ МОВИ В ТЕКСТ

Мова є найбільш природною формою людського спілкування, тому покращення якості її розпізнавання являє собою перспективний напрямок розвитку інтелектуальних систем управління. Звук характеризується показником частоти коливання на кожну одиницю часу. Цей показник може дуже відрізнятися для одного й того самого слова, вимовленого різними людьми: інший тембр голосу, різні інтонації, різна чистота вимови. Скільки людей, стільки й голосів. Голос – індивідуальна ознака особистості. Щоб навчити машину впізнавати мову, її потрібно змусити прослуховувати слова, сказані як однією людиною, так і різними людьми. Задача машини – прослухавши всі дані, взяти середні значення особливостей вимови, повністю виключити індивідуальність, щоб потім, почувши слово, не зробити помилку. Найбільші проблеми виникають в умовах: довільний користувач, спонтанна мова, яка супроводжується мовним «сміттям», наявність акустичних завад і скривлень. Щоб якість розпізнавання мови в таких умовах була прийнятною, треба налаштовувати параметри акустичної моделі під дані особливості.

2.1 Процес перетворення мови в текст

В первинному наближені, мова людини складається з фонем. Фонема є мінімальною одиницею, що впливає на зміст мови. Тобто, заміна однієї фонеми може привести до зміни змісту слова або фрази.

2.1.1 Представлення звукового сигналу для обробки

Мова – це безперервний аудіо потік, у якому різні мовні події плавно перетікають один в одного. Вона складається з послідовності звуків. Звук являє собою послідовність суперпозицій звукових коливань різних частот [8]. Звукові коливання характеризуються двома атрибутами – амплітудою та частотою коливань.

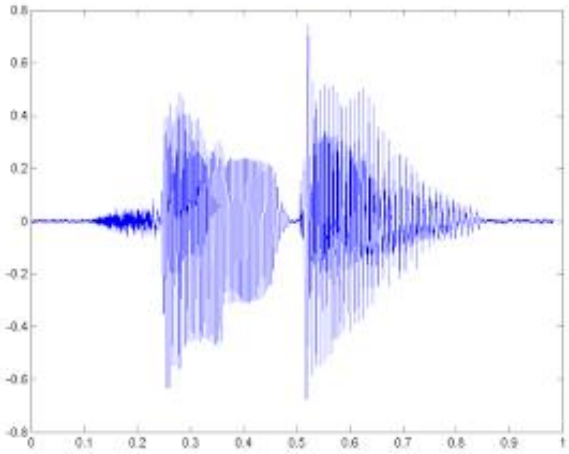

Рисунок 2.1 – Осцилограма слова Мама

Для перетворення звукової хвилі на цифровий сигнал, його необхідно розбити на безліч проміжків і взяти деяке усереднене значення на кожному з них.

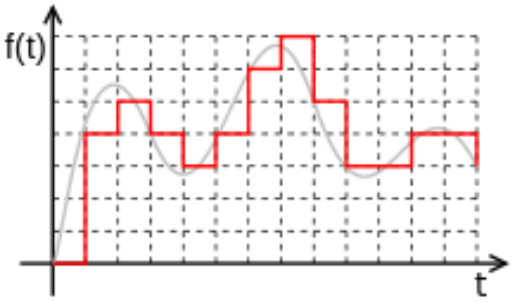

Рисунок 2.2 – Процес оцифровки звукового сигналу.

Таким чином набір коливань перетворюють в набір чисел, з якими можна маніпулювати на сучасних ЕОМ.

Задача розпізнавання мови полягає в порівняні множини чисел цифрового сигналу з множиною чисел, що описують кожне слово деякого словника.

2.1.2 Обробка звукового сигналу

Для можливості зіставлення звукового сигналу з мінімальною одиницею мови, фонемою, його необхідно поділити на безліч станів протяжністю від 10мс до 50 мс. У цій послідовності станів можна визначити більш-менш подібні класи фонем. Акустичні властивості форми сигналу, що відповідають одній і тій самій фонемі, можуть сильно відрізнятися в залежності від багатьох факторів - контексту фонеми, динаміка, навколишніх шумів тощо. Якщо взяти окрему ділянку сигналу і спробувати поставити у відповідність фонему, то складно це зробити однозначно. Багато фонем надзвичайно схожі одна на одну. Але якщо не можна дати однозначну відповідь, то можна міркувати в термінах ймовірностей: для даного сигналу одні фонеми більш вірогідні, інші менш, треті взагалі можна не розглядати. Тому кожен стан мовного потоку ділять на інваріанти, тобто набір векторних ознак, що описують кожен стан. Пошук таких інваріантів – завдання акустичного моделювання [8].

Основною одиницею мови є фонема. Але вона є досить крупною одиницею для того, щоб змоделювати мовний сигнал. Тому для обчислювальних цілей необхідно виділяти три окремих стани – початок, середина та кінець фонеми. Ці частини фонем складають фонетичний алфавіт всіх можливих звуків. Зазвичай їх називають сенонами. В акустичній моделі російської мови, яка використовувалась в даній роботі налічується близько 4000 сенонів. Завдяки сенонам описується все розмаїття звукового сигналу [8]. Отже, системи розпізнавання мови на вхід приймають звуковий потік, а на виході дає розподіл ймовірностей по кожному сенону.

У завданнях обробки мови існує кілька підходів до вилучення ознак. Основним з них є отримання мел-частотних кепстральних коефіцієнтів (Mel Frequency Cepstral Coefficients).

2.1.3 Отримання MFCC для алгоритмів розпізнавання

Насамперед вхідний сигнал розбивається на невеликі часові проміжки – фрейми. Причому фрейми повинні йти не строго один за одним, а так щоб сусідні фрейми частково перетиналися. Тобто кінець одного фрейму повинен частково накладатися на початок наступного. Для цього зазвичай звук нарізається ділянками по 25мс, а крок нарізки становить 10мс. Ці ділянки і називають фреймом.

Наступною задачею, яку необхідно вирішити при розпізнаванні мови, є розбиття даного мовного потоку на слова. Зазвичай в потоці мовлення є деякі паузи, які можна вважати роздільниками слів або фраз. У такому випадку нам потрібно знайти деяке значення, поріг - значення вище якого є словом, нижче - тишею. Це можна зробити, проаналізувавши ентропію. Ентропія – це міра невизначеності будь-якого стану. У нашому випадку ентропія вказує на те, як сильно змінюється наш сигнал в діапазоні заданого фрейму.

Щоб порахувати ентропію будь-якого фрейму треба виконати наступні дії [11]:

1) пронормувати сигнал так, щоб всі його значення лежали в діапазоні [-1;1];
2) побудувати гістограму значень сигналу на кожному фреймі. Це робиться за допомогою формули ентропії Шеннона:

$$E = \sum_{i=0}^{N-1} P[i] * \log_2(P[i]),$$

де P[i] це ймовірність настання і-го результату.

Отже, ми отримали характеристику фрейму, и тепер для того щоб виокремити звук від тиші, треба її порівнювати з якимось порогом. Зазвичай поріг ентропії беруть, як середнє між його максимальним і мінімальним значенням, але це не обов’язково.

Тепер в нас є набори фреймів, що відповідають кожному слову. Далі необхідно визначити чисельну характеристику кожного фрейму. Цією

характеристикою виступають MFCC. MFCC – це деяке уявлення енергії спектра сигналу. Плюси використання енергії спектра полягають в наступному [8]:

— спектр сигналу дозволяє проаналізувати хвильову природу сигналу;
— спектр розкладається на спеціальну mel-шкалу по необхідній кількості опорних точок. Це дозволяє виділити найбільш значущі частоти;
— кількість спектральних коефіцієнтів можна обмежувати будь-яким значенням. Це дозволить стиснути фрейм та кількість оброблюваної інформації;

Розглянемо більш детально спосіб обчислення MFCC коефіцієнтів для кожного фрейму. Представимо наш фрейм у вигляді вектора $X[k],\ 0 \leq k < N$, де N розмір фрейму. Розрахуємо спектр сигналу за допомогою дискретного перетворення Фур'є:

$$X[k] = \sum_{n=0}^{N-1} x[n] * e^{-2*\pi*i*k*n/N}, 0 \leq k < N,$$

де x[n] – послідовність вхідних фреймів, k – індекс ДПФ, n – часовий індекс вхідних фреймів, N – кількість фреймів вхідної послідовності.

Далі до отриманих значень необхідно застосувати віконну функцію Хеммінга. Це згладить значення векторів на кордонах фреймів:

$$H[k] = 0.54 - 0.46 * \cos(2 * \pi * k/(N-1)),$$

Тоді результатом буде вектор такого вигляду:

$$X[k] = X[k] * H[k], 0 \leq k < N,$$

Далі необхідно обчислити mel-фільтри. Mel – це психофізична одиниця висоти звуку, яка ґрунтується на суб'єктивному сприйнятті середньостатистичною людиною [12]. Вона залежить від частоти звуку, гучності і тембру. Іншими словами, це величина, яка значущість певної частоти. Перетворити частоту в mel можна за такою формулою (2.1):

$$M = 1127 * \log(1 + F/700),$$

де F – набір частот.

Зворотне перетворення виглядає так (2.2):

$$F = 700 * (e^{M/1127} - 1),$$

Графік залежності mel від частоти виглядає так (рисунок 2.3):

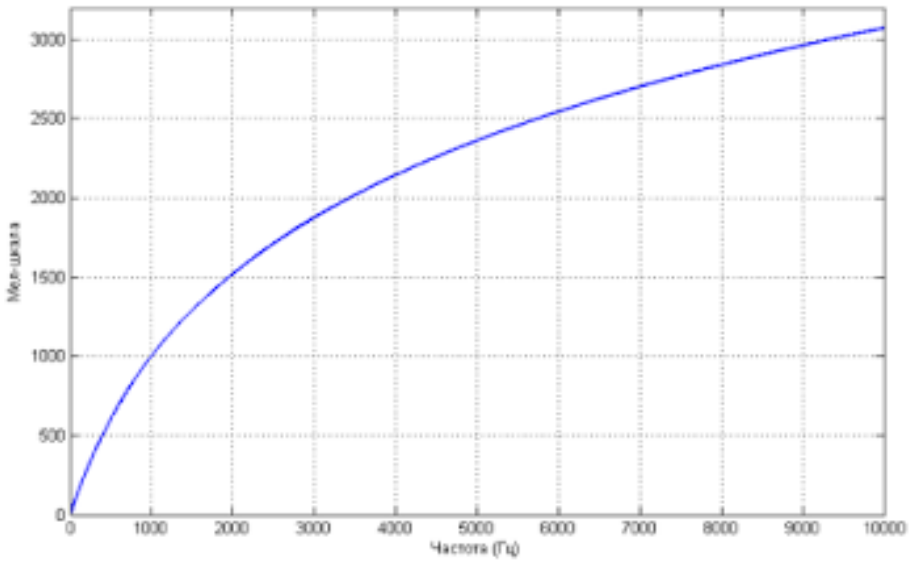

Рисунок 2.3 – Графік залежності mel від частоти

Для того, щоб розкласти даний спектр за mel-шкалою, необхідно буде створити набір фільтрів. По суті, кожен mel-фільтр дозволяє підсумувати кількість енергії на певному діапазоні частот і тим самим отримати mel-коефіцієнт. Необхідні нам фільтри легко знайти за такою формулою:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \le k \le f(m+1) \\ 0 & k > f(m+1) \end{cases},$$

де f[m] – опорні точки отриманого вище спектра. Вони рахуються по такій формулі:

f(m) = НЦЧ((fS+1) * h(m) / sampleRate),

де НЦЧ – найбільше ціле число, fS – довжина спектра, тобто кількість елементів на фрейм, sampleRate – частота звуку на даному фреймі (8000 hz, 16000 hz, 24000 hz і т.д.) , h(m) – шкала опорних частотних точок. Щоб її отримати треба:

1) Діапазон частот, в яких лежить людська мова, за формулою (2.1) перетворити в mel-шкалу, а потім рівномірно розбити на необхідну кількість опорних точок (зазвичай 12).

2) Далі за формулою (2.2) шкала переводиться назад у герци і отримуємо шукану шкалу опорних частотних точок.

Застосування фільтру полягає в попарному перемножуванні його значень зі значеннями спектра. Результатом цієї операції є mel-коефіцієнт. Оскільки фільтрів у нас M, коефіцієнтів буде стільки ж. Це робиться за формулою:

$$S[m] = \log(\sum_{k=0}^{N-1} |X[k]|^2 * H_m[k]), 0 \leq m < M$$

,

Після цього треба застосувати дискретне косинусне перетворення (DCT), для того щоб отримати ті самі кепстральні коефіцієнти. Його сенс полягає в тому, щоб зжати отримані результати, і тим самим підвищити значимість перших коефіцієнтів і зменшити значимість останніх. Формула DCT:

$$C[l] = \sum_{m=0}^{M-1} S[m] * \cos(\pi * l * (m + \frac{1}{2})/M)), 0 \leq l < M$$

,

де l – індекс кепстрального коефіцієнта.

Тепер для кожного фрейма ми маємо набір з M MFFC-коефіцієнтів, які можуть бути використані для подальшого аналізу.

2.1.4 Процес розпізнавання мови

В якості методу розпізнавання мови більшість сучасних систем використовують приховані марківскі моделі (HMM) . Використання HMM для розпізнавання мови грунтується на наступних припущеннях [9]:

• послідовні спостереження є статистично незалежними і, отже, ймовірність послідовності спостережень є просто добуток імовірності окремих спостережень;

• хоча мова являє собою нестаціонарний процес, вона моделює послідовністю векторів спостережень, які представляють собою кусочно-стаціонарний процес;

• власне марківське припущення, тобто припущення про те, що ймовірність перебування в деякому стані в момент часу t залежить тільки від стану, в якому процес знаходився в момент часу t -1. Найчастіше використовуються HMM з трьома станами (рисунок 3).

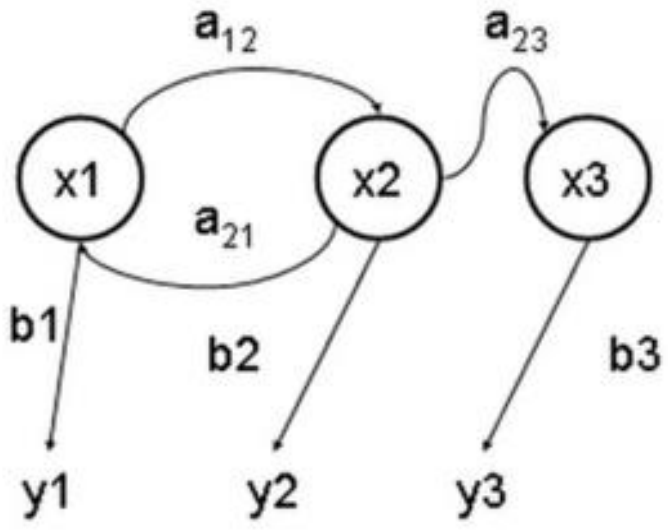

Рисунок 3 – Діаграма переходів в скритій марківській моделі
x — приховані стани
y — отримані результати
a — ймовірність переходів
b — ймовірність результатів

Кожний стан має імовірнісний розподіл серед всіх можливих вихідних значень. Модель представляє з себе марківський ланцюг, для якого нам відомі початкова ймовірність і матриця ймовірностей переходів. Прихованою вона називається тому, що ми не маємо інформації про її поточний стан. Прихована марківська модель являє собою кінцевий автомат, що змінює свій стан в кожен дискретний момент часу. Перехід з одного стану на інший здійснюється випадковим чином з деякою ймовірністю. В кожний момент часу, система породжує набір векторних ознак (фрейми з набором MFFC-коефіцієнтів) і, використовуючи існуючу ймовірнісну модель, намагається зіставити приховані стани з даним набором ознак [10]. Це завдання вирішується за допомогою методу Вітербі. Алгоритм Вітербі

дозволяє відновити найбільш ймовірну послідовність прихованих станів. Кожний прихований стан в задачі розпізнавання мови являє собою фонему мовної моделі. Для кожної фонеми архітектурою задається 3 стани, або вершини кінцевого автомата (початок фонеми, середина і кінець), в кожному з яких буде задано імовірнісний розподіл у вигляді суміші Гауссіан (ці параметри визначаються в процесі тренування). На етапі розпізнавання система буде оцінювати, на який розподіл (на яку частину якої фонеми) схожий поточний фрейм ознак. Стану прихованої марковської моделі задають тривалість фонем, а також порядок.

Алгоритм Вітербі дозволить нам отримати відповідь на питання, яка з гіпотез розпізнавання більш ймовірна. Тобто задача зводиться до того, щоб відновити найбільш ймовірну послідовність станів прихованої марківської моделі, яка могла б породити пред'явлений нам аудіо запис.

2.2 Проблеми систем розпізнавання мови

Починаючи з 1980-х років в розпізнаванні мови найгостріше стоїть проблема наявності перешкод. Системи ефективно працюють в ідеальних умовах записи, але при цьому не справляються з фоновими шумами. Штучні шуми виділити цілком можливо, але як відрізнити голос людини, який нам потрібно розпізнати, від голосу людини, що розмовляє по сусідству? Хоча проблема шумо стійкості до сих пір не вирішена, є кілька способів, які дозволяють її подолати [11].

По-перше, мовний корпус, на якому навчається акустична модель, штучно розширюють, додаючи до мовних сигналів з нього різноманітні перешкоди. Проблема тут в тому, що інший тип шумів, не врахований при навчанні, буде викликати збій в роботі.

По-друге, можна використовувати апаратні можливості підвищення шумо стійкості, що застосовувалися, наприклад, ще в перших моделях айфонів, які реалізовували функцію розпізнавання. У них використовувалися два мікрофони: перший, основний, ловив мову користувача разом з фоновими звуками, а другий

мікрофон на іншій стороні пристрою фіксував тільки фонові перешкоди. В результаті з сигналу, записаного першим мікрофоном, вичитали перешкоди, які були записані на другий мікрофон, і отримували досить чисту мову.

Друга проблема пов'язана з тим, що деякі акустичні характеристики користувача можуть не збігатися з типовими голосами, що використовуються в мовних корпусах. Стандартні алгоритми розпізнавання мови не дозволяють працювати з нестандартною промовою. Проблема особливо актуальна при розпізнаванні мови на нерідній для диктора мові.

Один з варіантів подолання зазначеної проблеми полягає в накопиченні великої кількості нестандартних фрагментів мови, з різними акцентами, в навчанні системи на основі розширеного корпусу. Але такий спосіб потребує великої кількості акустичних даних для навчання.

Ще одне рішення можливе за допомогою налаштування і адаптації стандартної акустичної моделі на голос конкретного користувача. Цей спосіб полягає в тому, що користувач або в процесі експлуатації, або на попередньому етапі навчання акустичної моделі читає певний текст. Після цього на основі цих аудіо даних змінюються параметри акустичної моделі, щоб врахувати варіації голосу користувача. Часто адаптації буває досить, щоб система стала надійніше розпізнавати голос користувача, навіть при наявності в мові дефектів.

Оскільки сучасні системи розпізнавання мови дуже ресурсомісткі і вимагають високої обчислювальної потужності, щоб розпізнавати мову в режимі реального часу, більшість сучасних систем реалізуються або на стаціонарних комп'ютерах, або в клієнт-серверному режимі, коли мобільний пристрій записує мовний сигнал, відправляє його по інтернету на спеціальний сервер і отримує у відповідь транскрипцію. Такі системи не дуже зручні, тому що в ряді випадків передача мови на сервер неможлива через відсутність інтернету або в зв'язку з підвищеними вимогами до безпеки. І хоча зараз досягнута хороша точність розпізнавання, яку можна порівняти з людською, але для малопродуктивних автономних пристроїв, в яких до того ж неприйнятні великі витрати енергії, доводиться різко спрощувати

моделі, що призводить, в свою чергу, до зниження точності. Тому сучасні дослідження ведуться в області автономного розпізнавання мови, в яких весь процес відбувається на мобільних пристроях. Якщо вирішити це завдання і спростити алгоритми розпізнавання мови без значущих втрат в точності, то використовувати розпізнавання мови можна буде повсюдно.

2.3 Де потрібні технології розпізнавання мови

Перш за все, технології розпізнавання мови використовуються для голосового набору команд, в ситуаціях, при яких говорити набагато простіше, ніж друкувати. Розпізнавання мови застосовується в системах інтерактивного мовного самообслуговування, коли, наприклад, на телефонні дзвінки в компанії відповідає робот, який може розібратися зі стандартними питаннями з області підтримки. Ще одне застосування технологій розпізнавання голосу - диктування текстів. Нарешті, все частіше з'являються системи з голосовим управлінням будь-якою технікою, наприклад «розумний дім», або автомобілем. Область застосування буде найближчим часом безперервно розширюватися в зв'язку з безсумнівною зручністю для користувача голосових команд та прогресом в точності розпізнавання мови.

2.4 Висновки до розділу

У даному розділі було розглянуто проблематику систем розпізнавання мови та алгоритмів розпізнавання.

Було детально розглянуто алгоритм перетворення вхідного звукового сигналу у текст. Також розглянуто основний алгоритм, що використовується в більшості систем розпізнавання мови – алгоритм прихованих Марківських моделей.

Було виділено основні проблеми сучасного розпізнавання мови. До них відносять: проблеми наявності перешкод, відмінність деяких характеристик користувача від основних, на яких навчалася акустична модель. Але найважливіша

проблема полягає в необхідності значних обчислювальних та ресурсомістких характеристик системи. І, хоча сучасні системи розпізнавання мови, які ґрунтуються на клієнт-серверної взаємодії, дають значні показники точності в 90%, проте для малопродуктивних автономних пристроїв точність розпізнавання залишається доволі низькою. Для підвищення якості розпізнавання таких систем, основним способом є адаптацію наявних не великих акустичних даних для конкретного користувача.

3. ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ РОЗПІЗНАВННЯ МОВИ ТА АДАПТАЦІЇ АКУСТИЧНОЇ МОДЕЛІ

3.1 Класифікація систем розпізнавання мови

На сьогоднішній момент вчені виділяють кілька ознак, за якими класифікуються процеси перетворення людської мови в цифрову інформацію. За цими ознаками можна кваліфікувати дану систему. Нижче описані і розглянуті основні ознаки [12].

1. Розмір словника:

— малий словник - близько 100 слів;

— середній словник - близько 1000 слів;

— великий словник - близько 5000 і більше.

При великому словнику існує велика ймовірність отримання помилок при розпізнаванні. Таким чином, словник складається з 10 цифр, які між собою не співзвучні. Такий словник буде розпізнано практично без помилок. А якщо вибрати словник з декількох тисяч слів ймовірність помилки може досягати 50%. У додаванні до цього, якщо в словнику є дуже багато омофонів, то точність розпізнавання може зменшиться практично до 0%.

2. Залежність системи:

— дикторозалежні;

— дикторонезалежні.

Дикторозалежні системи призначені для користування однією людиною. Цю систему можна налаштувати на параметри конкретного диктора. У таких системах частота помилок менше в 3-5 разів в порівнянні з дикторонезалежної системами, тому що алгоритм зводиться до навчання системи розпізнавання на основі одного

голосу. Друга система реалізується набагато складніше, тому що система повинна пристосовуватися до особливостей різних дикторів, їх темпу і тембру мови, а також ритму і такту. Ця система розроблена для користування абсолютно різними людьми. При розпізнаванні мови з акцентом або діалектом точність зменшується в 2-4 рази.

3. Тип мови [12]:

— злита мова;

— зв'язкові слова і фрази;

— окремі слова.

Зазвичай слова в нашій мові розділені між собою невеликим ділянкою тиші. Якщо ця тиша присутній, то мова роздільна. В іншому випадку, мова злита. Злита мова - це звичайна повсякденна мова. Процес розпізнавання мови здійснюється набагато важче, так як слова не мають чітких меж і звуки на стику слів сильно змащені.

4. Якість розпізнається мови:

— чиста мова;

— слабо зашумлена мова;

— сильно зашумлена мова.

Під чистою промовою розуміється сигнал, що містить тільки мову без будь-яких перешкод. На практиці не існує ідеально чистих аудіо даних. Існують різні алгоритми з придушення шуму, проте сильно зашумлену мову дуже складно розпізнати.

5. Спосіб розбиття мови на елементарні одиниці:

— по фонем;

— по частинах слів;

— по словам.

Мова людини формується шляхом вимови окремих звуків, при об'єднанні яких утворюються осмислені слова і пропозиції. При створенні системи розпізнавання мови також можна виділити неподільні елементарні одиниці, як фонеми, склади, слова, або ж задати їх штучно.

6. Положення пристрою захоплення звуку:

- близьке розташування (до 20 см);

- далеке розташування (більше 20 см);

Важливу роль в розпізнаванні грає відстань між диктором і пристроєм захвату мови, так як ця відстань впливає на ступінь присутності стороннього шуму.

3.2 Архітектура систем розпізнавання мови

Типова архітектура статистичних систем автоматичної обробки мови така [12]:

— модуль шумоочистки і виділення корисного сигналу.

— акустична модель - дозволяє оцінити розпізнавання мовного сегмента з точки зору схожості на звуковому рівні. Для кожного звуку спочатку будується складна статистична модель, яка описує проголошення цього звуку в мові.

— мовна модель - дозволяють визначити найбільш ймовірні словесні послідовності. Складність побудови мовної моделі багато в чому залежить від конкретної мови. Так, для англійської мови, досить використовувати статистичні моделі (так звані N-грами). Для високофлективних мов (мов, в яких існує багато форм одного і того ж слова), до яких відноситься і російська, мовні моделі, побудовані тільки з використанням статистики, вже не дають такого ефекту – занадто багато потрібно даних, щоб достовірно оцінити статистичні зв'язки між словами. Тому застосовують гібридні мовні моделі, які використовують правила російської мови, інформацію про частини мови і форми слова і класичну статистичну модель;

— декодер – програмний компонент системи розпізнавання, який поєднує дані, одержувані в ході розпізнавання від акустичних і мовних моделей, і на підставі їх об'єднання, визначає найбільш ймовірну послідовність слів, яка і є кінцевим результатом розпізнавання злитого мовлення.

Етапи розпізнавання

Обробка мови починається з оцінки якості мовного сигналу. На цьому етапі визначається рівень перешкод і спотворень.

Результат оцінки надходить в модуль акустичної адаптації, який управляє модулем розрахунку параметрів мови, необхідних для розпізнавання.

У сигналі виділяються ділянки, що містять мова, і відбувається оцінка параметрів мови. Відбувається виділення фонетичних і просодичних імовірнісних характеристик для синтаксичного, семантичного та прагматичного аналізу. (Оцінка інформації про частини мови, формі слова і статистичні зв'язки між словами.)

Далі параметри мовлення надходять в основний блок системи розпізнавання – декоде. Це компонент, який зіставляє вхідний мовний потік з інформацією, що зберігається в акустичних і мовних моделях, і визначає найбільш ймовірну послідовність слів, яка і є кінцевим результатом розпізнавання.

3.3 Сучасні системи розпізнавання мови

Технологія розпізнавання мови є вкрай популярною і перспективною, тому існує безліч систем, що реалізовують її. Всі існуючі акустичні моделі можна поділити на дві категорії: з відкритим вихідним кодом і системи з закритим вихідним кодом. Реалізація систем розпізнавання мови з закритим вихідним кодом виконана більш якісно, такі системи мають високу точність розпізнавання мови. Проте вони можуть не мати необхідної документації, пов'язаної з інтеграцією інших рішень в свою роботу. Найвідоміші з них це: Dragon Mobile SDK, Google Speech Recognition API, Yandex Speech Kit, Microsoft Speech API [2]. Часто подібні системи є платними, тобто доведеться купувати ліцензію, щоб використовувати мовні технології, представлені компанією-розробником. Системи розпізнавання мови з відкритим вихідним кодом зазвичай безкоштовні, але для їх ефективної роботи необхідно створення вихідної бібліотеки з розширеним набором артикуляційних даних, мовних одиниць і т.д. Це означає, що за розвиток таких відкритих систем відповідальні ті, хто ними користується. Серед акустичних моделей з відкритим

вихідним кодом можна виділити: CMUSphinx, Kaldi, HTK, Julius. Такі системи є більш доступні для використання, інтегрування нових можливостей та проведення досліджень [2].

Але найважливішою відмінністю систем розпізнавання мови є спосіб реалізації. Більшість сучасних систем, які повсюдно використовуються, реалізуються в клієнт-серверному режимі. В цьому випадку записаний мовний сигнал відправляється по інтернету на спеціальний сервер, там він обробляється, і відправляє у відповідь готову транскрипцію. Такі системи можуть працювати з більш ресурсномісткими моделями і надають високі обчислювальні можливості. Точність розпізнавання таких систем дуже висока і досягає 90% точності розпізнаних слів. Проте в ряді випадків передача мови на сервер неможлива через відсутність інтернету або в зв'язку з підвищеними вимогами до безпеки. В цьому використовують системи, в яких весь процес відбувається на самому мобільному приладі. Для таких приладів доводиться різко спрощувати моделі, що призводить, в свою чергу, до зниження точності. Тому сучасні дослідження ведуться в області автономного розпізнавання мови, в яких весь процес відбувається на мобільних пристроях.

Для даної роботи було обрано систему CMUSphinx з декількох причин, основними з яких є:

1) Наявність великої кількості документації по користуванню;
2) Ця система реалізована без клієнт-серверної архітектури, і призначена для використання на малопродуктивних автономних пристроїв
3) Наявність великої кількості натренованих акустичних і мовних моделей для різних мов.

3.4 Огляд інструментальної системи CMUSphinx

На даний момент актуальною є версія sphinx4-5prealpha [13]. Sphinx-4 представляє собою модульний фреймворк (рисунок 1). Модульна структура

дозволяє варіювати параметри системи виходячи з вимог конкретного завдання. виділяються 3 основних модуля: FrontEnd, Decoder і Linguist [14]. FrontEnd перетворює вхідні дані в вектор параметрів. Linguist на основі обраної мовної, акустичної моделей і словника будує SearchGraph. Нарешті, подмодуль Decoder'a – SearchManager - використовує вектор параметрів і побудований граф для декодування і видає результат.

Рисунок 1 - Взаємодія модулів системи CMUSphinx

Блок Front End відповідає за збір, анотування і обробку вхідних даних. Крім того, він витягує об'єкти з вхідних даних для читання за допомогою декодера.

База знань містить інформацію необхідну для декодера. Ця інформація включає в себе акустичну модель і модель мови. База знань також може отримати відповідь від декодера, що дозволяє базі знань динамічно змінюватися на основі результатів пошуку. Ці модифікації можуть включати в себе перемикання акустичної моделі та мовної моделі, а також оновлювати параметри, такі як

дисперсія перетворення для акустичних моделей.

Декодер виконує більшу частину роботи. Він зчитує дані з Front End, зіставляє їх з даними з бази знань і відгуком програми і виконує пошук найбільш ймовірних послідовностей слів, які були представлені рядом особливостей.

На відміну від безлічі архітектур розпізнавання мови, Sphinx4 дозволяє додатку контролювати безліч функцій мовного движка. Під час декодування, додаток може отримувати дані від декодера, в той час, коли він здійснює пошук. Ці дані дозволяють додатком відстежувати, як відбувається процес декодування і також дозволяє додатку впливати на процес декодування до його завершення. Крім того, додаток може оновлювати базу знань в будь-який час.

3.5 Висновки до розділу

В даному розділі була розглянута класифікація систем розпізнавання мови, описана їх структура. Був проведений огляд існуючих систем розпізнавання мови та обрана найбільш актуальна для адаптації система розпізнавання мови.

Вибір системи CMUSphinx був обґрунтований насамперед тим, що процес розпізнавання мови реалізований безпосередньо на тому приладі, на який вона встановлена. Такі системи є більш мобільні, але через брак обчислювальних ресурсів, точність їх розпізнавання досить низька. Тому вони потребують додаткового проведення адаптації для конкретних умов.

Також було більш детально розглянуто алгоритм розпізнавання мови в системі CMUSphinx та описано її архітектуру.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Структура програми

Система для адаптації акустичної моделі була написана за допомогою мови програмування Java та з використанням JavaFX. Для розпізнавання мови було використано програму Pocketsphinx [15, 16] з використанням бібліотеки Sphinxbase [17]. Для проведення адаптації використовувалися інструменти тренування акустичної моделі Sphinxtrain [18].

Інтерфейс програми має наступний вигляд (рисунок 4.1):

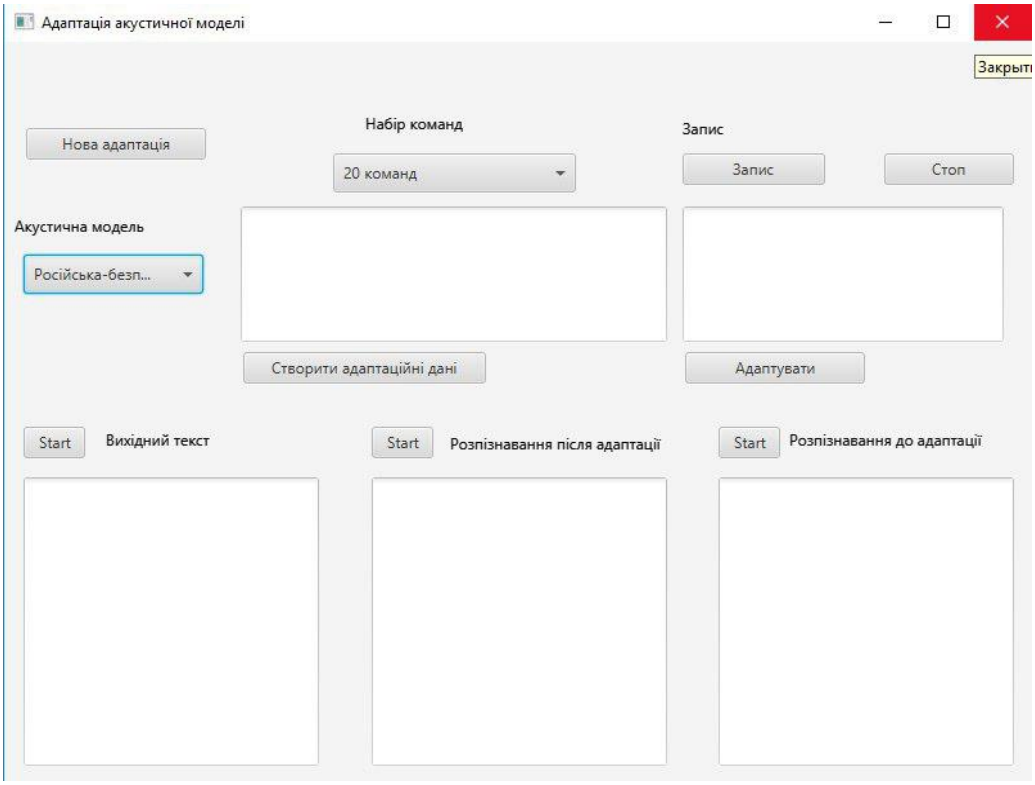

Рисунок 4.1 – Інтерфейс програми адаптування акустичної моделі

Він надає можливість працювати з трьома основними етапами адаптації:

1) Створення адаптаційних даних;

2) Адаптування параметрів акустичної моделі на основі цих даних;

3) Порівняння результатів розпізнавання мови до та після адаптації.

Діаграмма залежностей класів програми має такий вид (рисунок 4.2)

Рисунок 4.2 – Діаграма залежностей класів програми адаптації акустичної моделі

Структура програми складається з одного пакету sample, який містить 8 класів. Класи Model і ModelDAO, та exampleFile і exampleFileDAO були створенні за DAO Pattern (data access object), що використовується для абстрагування і інкапсуляції доступу до джерела даних [19]. В даному випадку абстрагувався доступ до колекцій з відповідними об’єктами Model та exampleFile, що описують файли доступних акустичних моделей та доступних наборів команд. Клас RecFile описує об’єкт, що містить данні про формат аудіо файлів. Клас Main є точкою входу програми, в ньому ініціалізується головне вікно програми та запускається Controller. Клас Controller є головним класом програми, в якому описаний весь функціонал.

Клас CaptureThread описує потік в якому можливий запис та збереження аудіо файлів.

4.2 Робота користувача з програмою

Перш за все, користувач повинен обрати зі списку акустичну модель для адаптації (рисунок 4.3) та натиснути на кнопку «Нова адаптація». Це завершить всі поточні сеанси адаптації (якщо вони є) та створить нову папку, в якій буде файл обраної акустичної моделі. Далі всі адаптаційні дані та власне адаптована акустична модель буде знаходитись в цій папці (рисунок 4.4).

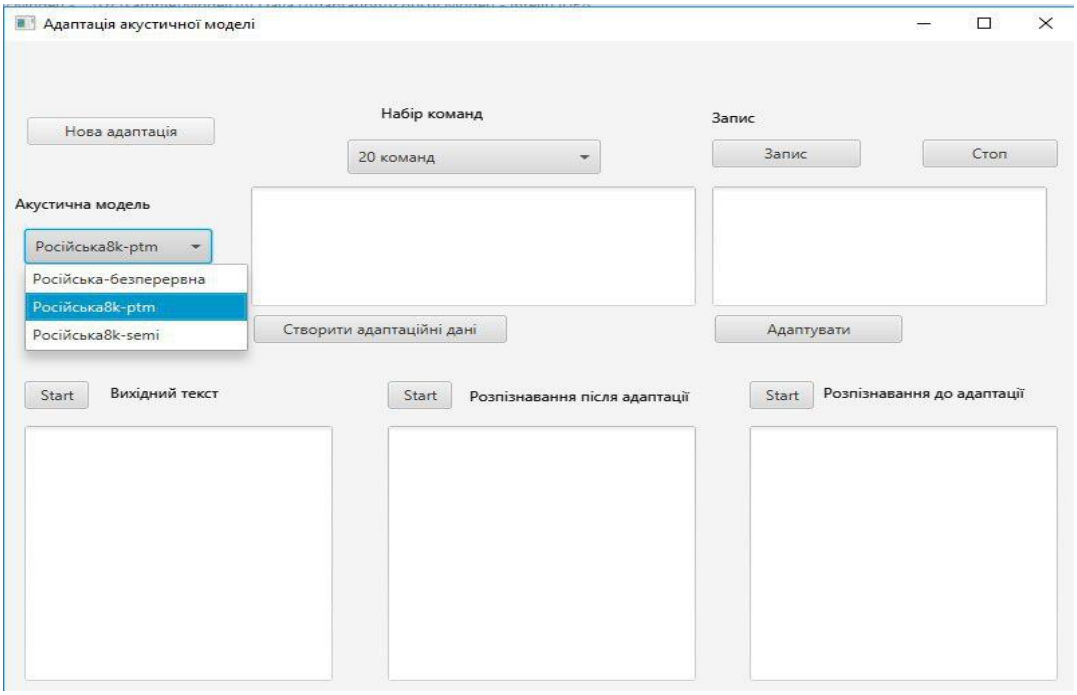

Рисунок 4.3 – вибір акустичної моделі

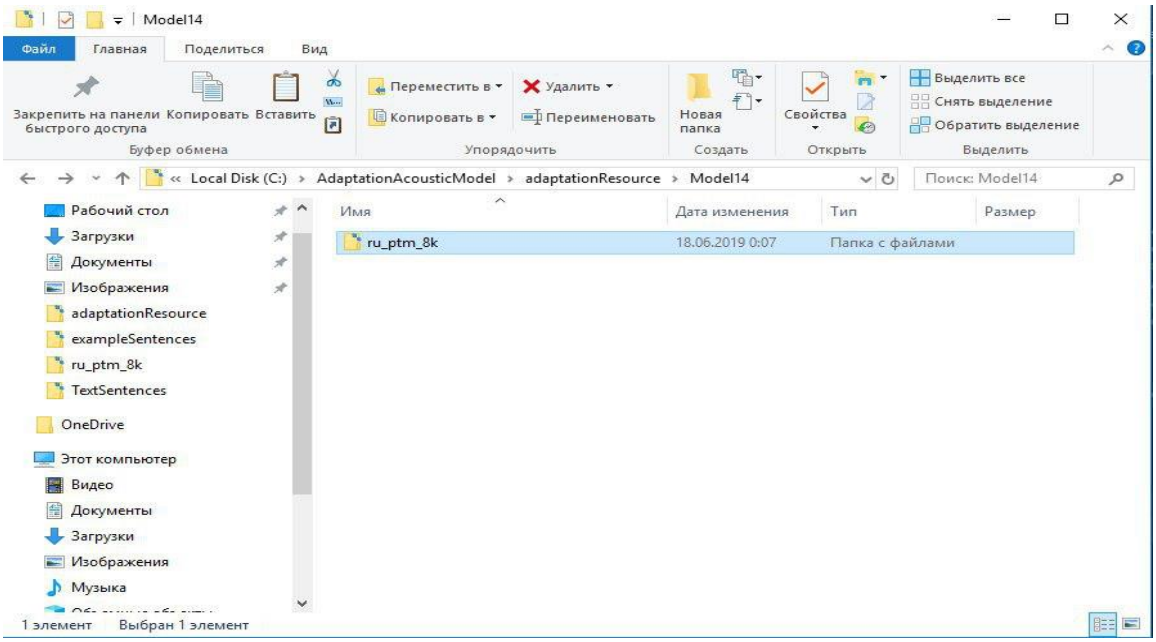

Рисунок 4.4 – створена папка для даних адаптації

Далі необхідно створити набір команд для адаптації чи обрати зі списку існуючі.

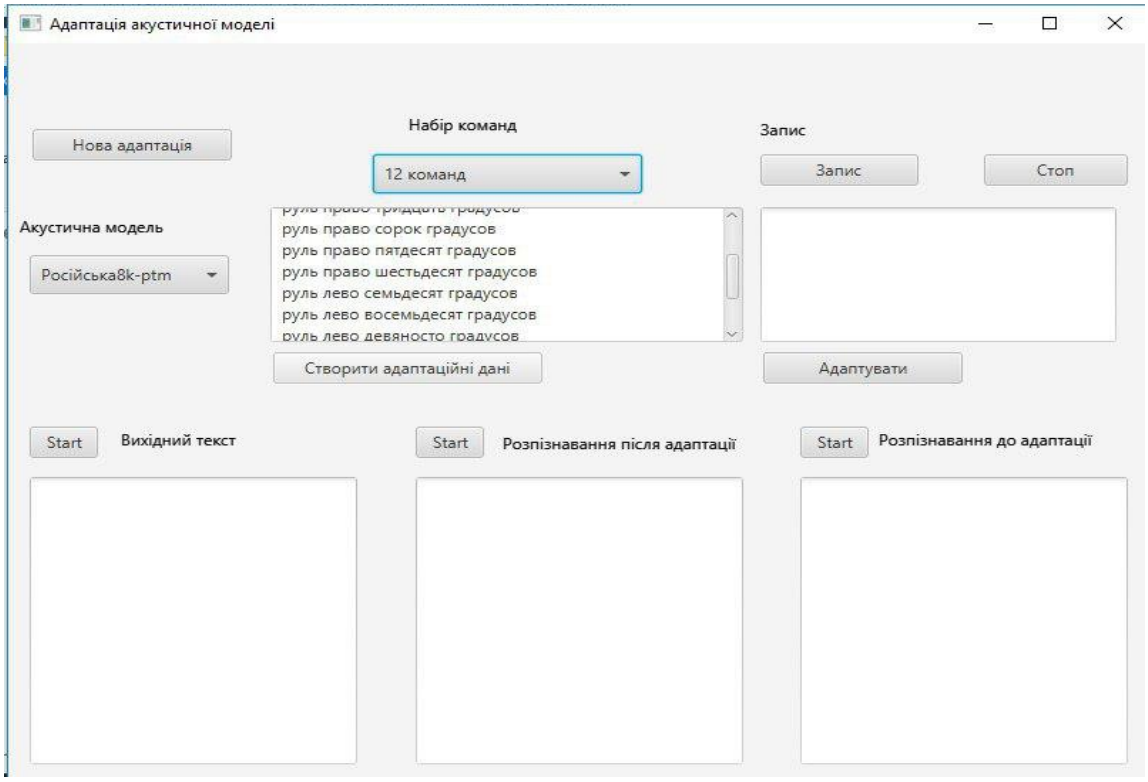

Рисунок 4.5 – Створення набору команд

Після цього треба натиснути кнопку «Створити адаптаційні дані». Система відповідно до створених команд та обраної акустичної моделі створить файли мовної моделі, словника, файл транскрибування, файл .fileids, та всі інші необхідні для проведення адаптації файли (рисунок 4.6):

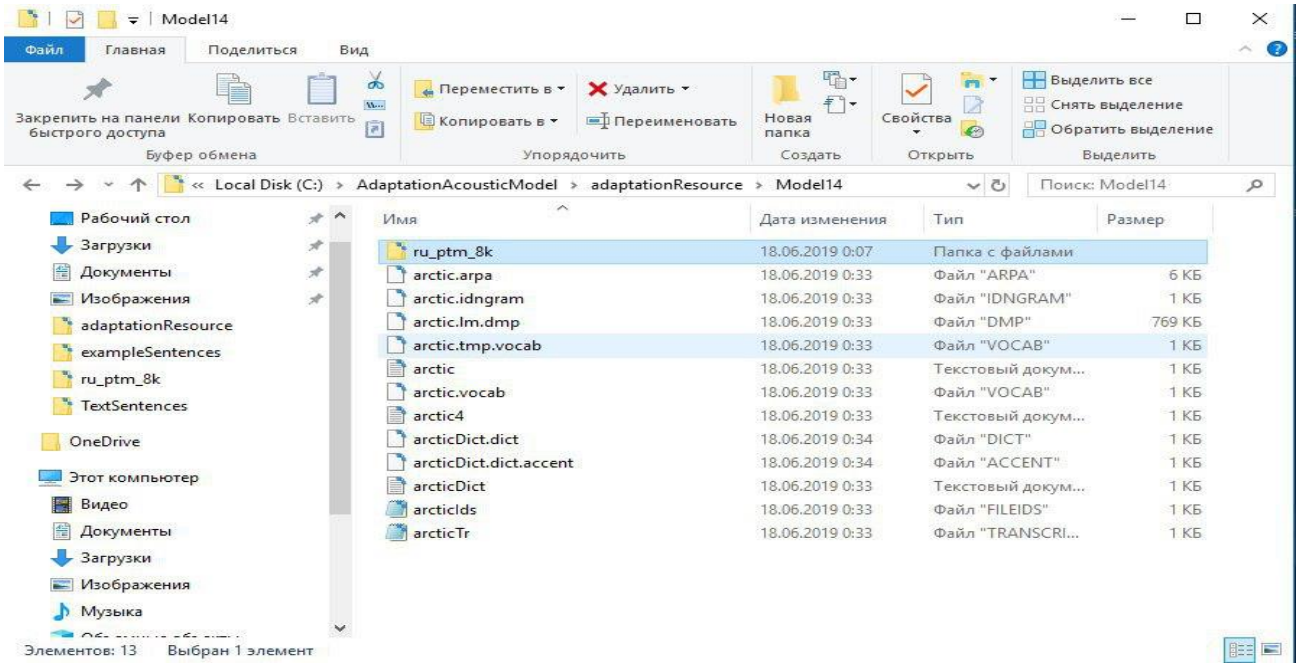

Рисунок 4.6 – Створенні файли

Далі необхідно створити один аудіо запис для кожної команди. Треба просто натиснути кнопку «Запис» вимовити команду та натиснути кнопку «Стоп». Таким чином створюються всі аудіо записи кожної команди.

Після того, як користувач вимовить всі команди, треба натиснути кнопку «Адаптувати». Система створить файли акустичних об’єктів для кожного аудіо запису та проведе MLLR адаптацію. Результатом адаптації буде файл mllr_matrix (рисунок 4.7), який являє собою набір трансформацій, що зменшать розходження між акустичною моделлю та адаптивними даними.

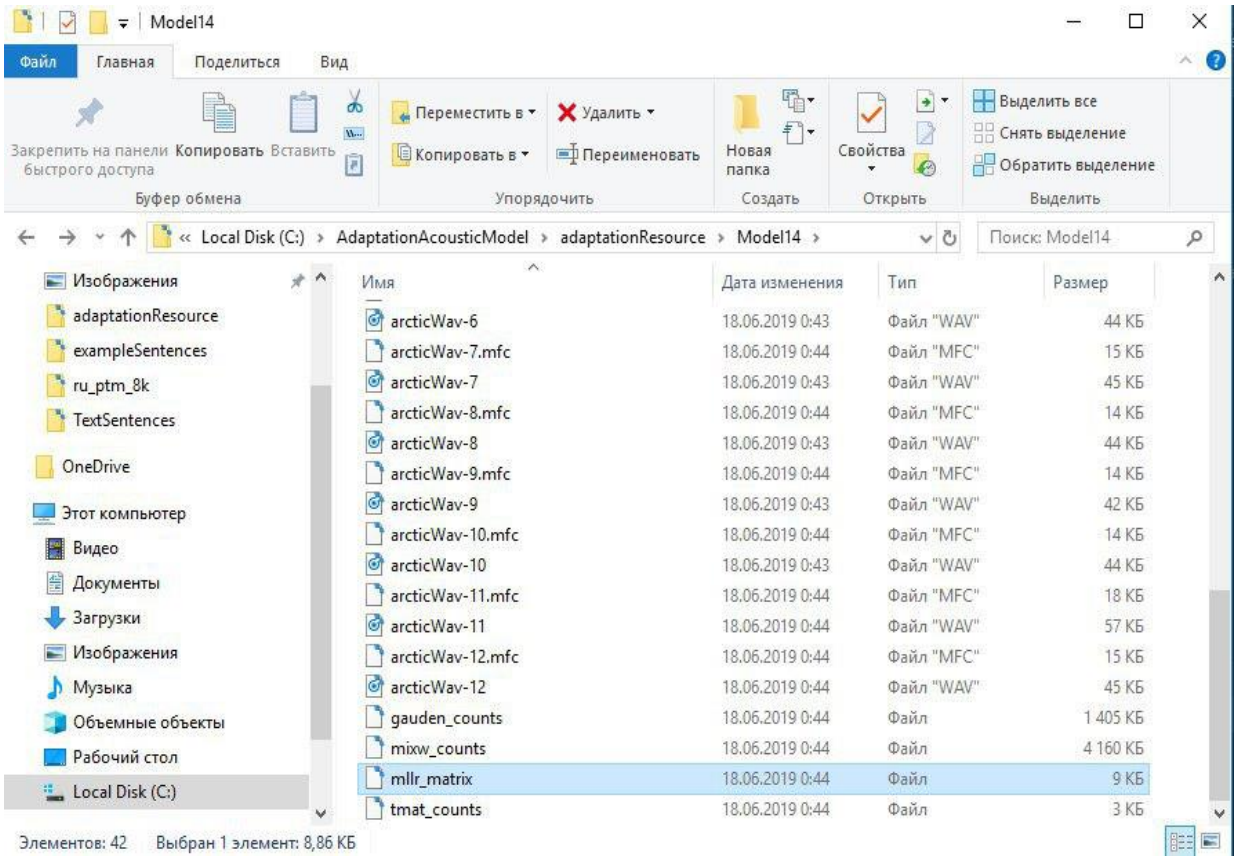

Рисунок 4.7 – Створений mllr_matrix файл

Далі залишається лише порівняти результат розпізнавання команд за використанням початкової акустичної моделі та адаптованої (рисунок 4.8):

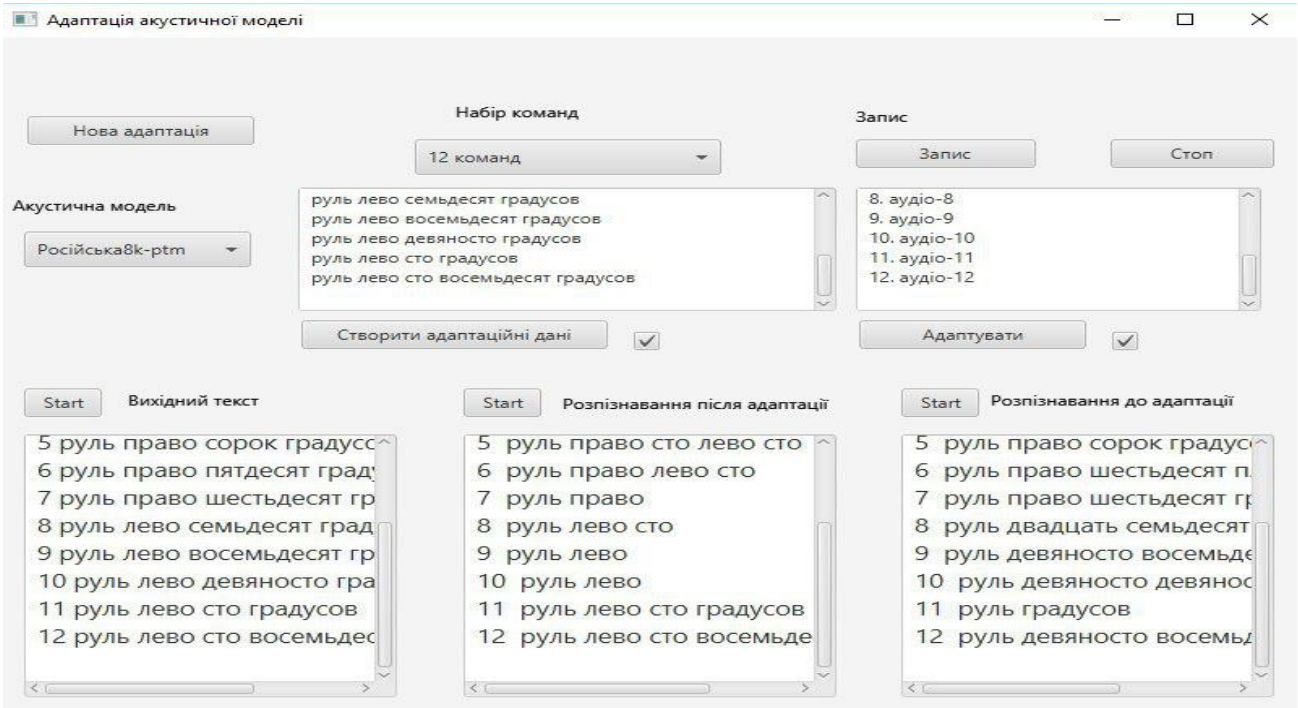

Рисунок 4.8 – результат розпізнавання команд

4.3 Алгоритм адаптації

Початкові акустичні моделі здебільшого не підходять для використання в будь-яких умовах навколишнього середовища. Точність розпізнавання таких моделей дуже низька. Для вирішення цієї проблеми існує методика адаптації акустичної моделі до певного середовища (набір обладнання, диктор, зовнішні шуми). Ця методика дозволяє поліпшити точність розпізнавання мови до певного диктора.

Ідея методу полягає в тому, щоб акустична модель краще підходила до набору даних, що використовуються для адаптації. Отже, дані повинні мати пряме відношення до сфери розпізнавання мови. Ми можемо адаптуватися до конкретного середовища запису і передавального пристрою (нашому мікрофона) або диктора. Процес адаптації вимагає записаних даних та покращує вже наявну модель. Досить мати 5 хвилин запису мови, щоб поліпшити точність в розпізнаванні конкретного диктора. Розглянемо етапи адаптації.

— Перший етап - створення адаптаційного набору даних. Адаптаційний набір даних складається з наступних частин:
  - набір фраз;
  - словник, що містить вимова всіх слів цих виразів;
  - звукові записи фраз набору. Звукові записи повинні мати формат .wav, частоту 8000 Гц і бути записані на одному каналі в монорежимі.

— Наступний етап - створення файлів, що містять ознаки акустичної моделі на основі звукових записів з набору. Файли ознак мають формат mfc.

— Далі необхідно зібрати статистику з адаптаційних даних.

— Останній крок - зміна акустичної моделі за допомогою зібраних

— даних.

— Найбільш популярні 2 методу проведення адаптації:

— MLLR-трансформація.

Даний метод полягає в обчисленні набору трансформацій, які зменшать розбіжність між моделлю і адаптаційними даними. Це перетворення зміщує середні значення моделей фонем і змінює їх дисперсію так, щоб прихована марківських модель генерувала дані, більш наближені до адаптаційних.

MAP адаптація.

У цій адаптації параметри моделі перевизначаються індивідуально. Для кожного параметра прихованої марківської моделі обчислюється оцінка максимальної правдоподібності на основі адаптаційної дати. Потім формується MAP оцінка шляхом зсуву попередніх параметрів до оцінки максимальної правдоподібності. Іншими словами, MAP оцінка це зважене середнє між оцінкою максимальної правдоподібності і попередньої оцінкою. В оцінці максимальної правдоподібності (ML estimation) правдоподібність тренувальних даних p (x | λ) максимізувало оцінкою параметра λ. MAP у чому схожий на ML оцінку, але на додаток до правдоподібності додається інформація про попередні параметрах, тобто максимізується p (x | λ) g (λ), де g (λ) це дані до адаптації. На практиці в якості даних до адаптації використовується або акустична модель, адаптована SI методом, або модель адаптується інтерактивно, і ми використовуємо параметри попередніх ітерацій. При використанні попередніх даних, для оцінки потрібно набагато менше даних, що є великою перевагою, коли набір адаптаційних даних малий. MAP вимагає більше адаптують даних, ніж MLLR, щоб досягти тієї ж ефективності, так як кожен параметр обчислюється окремо. Якщо адаптаційні дані складаються з одного виразу, то вираз містить тільки малу частину всіх можливих Трифонів. Параметр оновлюється тільки, якщо Трифон з'явився в адаптаційних даних. Чим більше разів Трифон з'являється в даних, тим більше ваги має оцінка максимального правдоподібності. Виходячи з цього, якщо адаптаційний набір малий, то параметри моделі зміняться не дуже сильно, так як нова оцінка не матиме достатньої ваги. MLLR працює краще ніж MAP на малих наборах, так як матриця трансформації може бути обчислена навіть з одного виразу. MAP адаптація перевершує MLLR, коли набір даних стає більше, так як при великому наборі даних, вага оцінки

правдоподібності буде відчутно більше, ніж у параметрів до адаптації. В цьому випадку MAP очікування буде ближче до оцінці правдоподібності, обчисленої на адаптаційних даних. У даній роботі був обраний метод MLLR адаптації, так як адаптаційний набір даних недостатній для MAP трансформації.

4.4 Результати адаптації

Завданням адаптації є покращення точності розпізнавання мови. Проте може так статися, що після адаптації результат або не зміниться, або навіть погіршиться. Це відбувається, коли умови при яких проводилась адаптація не співпадають з умовами використання системи. Метою дослідження було порівняти якість розпізнавання мови адаптованої та не адаптованої акустичної моделі.

Для початку необхідно створити тестові дані, однакові для всіх умов дослідження. Набір тестових даних для дослідження складається з наступних 30 команд:

- Встреча с заказчиками
- Подготовить отчет
- Подготовить отзыв
- Совещание о состоянии работ по проекту
- Написать объявление
- Открыть програму
- повернуть направо десять градусов
- повернуть направо пятнадцать градусов
- повернуть направо двадцать градусов
- повернуть направо тридцать градусов
- повернуть направо сорок градусов
- повернуть направо пятдесят градусов
- повернуть направо шестьдесят градусов
- повернуть налево семьдесят градусов

- повернуть налево восемьдесят градусов
- повернуть налево девяносто градусов
- повернуть налево сто градусов
- повернуть налево сто восемьдесят градусов
- двигаться прямо
- двигаться назад
- выключить компьютер
- перейти в спящий режим
- удалить програму
- очистить корзину
- приготовить данные для адаптации
- посчитать стоимость заказа
- открыть дверь
- закрыть окно
- перейти в панель задач
- очистить историю браузера

Тестування проводилося в різних умовах зашумленності:

- В умові постійного механічного шуму (звуку пилососа);
- В умові чутливості мікрофона 12дб, без сторонніх;
- В умові чутливості мікрофона 24дб, але без сторонніх шумів;
- В умові мінливого механічного шуму.

Тести проводились на безперервній акустичній моделі, оскільки вона найкраще підходить для MLLR адаптації.

Для визначення ефективності розпізнавання мови використовувалася метрика Word Error Rate (WER) [20], заснована на підрахунку вставок, замін і вилучень слів. Нехай в тексті, отриманому в результаті розпізнавання мови, N слів. Тоді WER =(S + D + I)/N , де

• S - кількість замінених слів

• D - кількість вилучених слів

• I - кількість вставлених слів

Тобто, чим вище WER, тим менш якісне розпізнавання.

Результати дослідження показанні в таблиці 4.1:

	До адаптації	З адаптованою моделлю	Приріст точності
Постійний шум	36,4%	31,2%	5,2%
Чутливість мікрофону 12дб	34,4%	32,4%	2%
Чутливість мікрофону 24дб	39,2%	38,4%	0,8%
Мінливий шум	43,6%	37,2%	6,4%

Таблиця 4.1 – Результати експерименту

4.5 Висновки до розділу

В даному розділі розглянута структура програми адаптації, описано спосіб взаємодії користувача з системою. Також розглянуто використовуваний алгоритм адаптації. Оцінено якість роботи алгоритму розпізнавання для безперервної акустичної моделі в різних умовах шуму до та після адаптації.

Розроблено методику експерименту, в ході якого було встановлено оцінки якості роботи алгоритму. Алгоритм протестований на тестових даних. Було встановлено, що проведення адаптації акустичної моделі призводить до підвищення ефективності розпізнавання мови.

ВИСНОВКИ

Дана робота була присвячена огляду систем та алгоритмів розпізнавання мови та створення системи, що надає можливість адаптації акустичних моделей.

Програму можна використовувати для покращення якості розпізнавання мови в різних умовах навколишнього середовища.

Додаток було написано на мові програмування Java з використанням графічного інтерфейсу JavaFX. Для розпізнавання мови було використано програму Pocketsphinx з використанням бібліотеки Sphinxbase. Для проведення адаптації використовувалися інструменти тренування акустичної моделі Sphinxtrain.

В ході роботу було проаналізовано ефективність роботи системи CMU Sphinx, та була покращена ефективність розпізнавання за рахунок адаптації акустичної моделі та створення більш компактного словника.

Для роботи з даним програмним забезпеченням необхідний лише комп’ютер середньої потужності.

Користувач має змогу власноруч адаптувати існуючу акустичну модель під свій голос та умови навколишнього середовища.

Середнє значення WER для безперервної акустичної моделі становить 38,6% для не адаптованої акустичної моделі та 34,8% після адаптації. Зі збільшенням словника якість розпізнавання суттєво падає. Для словника обсягом в 20000 слів WER становить майже 90%. Для цього в програмному модулі було реалізовано вибір найбільш оптимального словника виходячи з адаптаційних даних.

Було встановлено, що проведення адаптації акустичної моделі призводить до підвищення ефективності розпізнавання мови в середньому на 3,8%.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Технологии распознавания речи. [Електронний ресурс] – Режим доступу: https://postnauka.ru/faq/82575
2. Обзор систем распознавания речи. [Електронний ресурс] – Режим доступу: https://nauchforum.ru/studconf/tech/xliii/18095
3. Placeway, Chen, Eskenazi, Jain, Parikh, Raj, Ravishankar, Rosenfeld, Seymore, Siegler, Stern, Thayer. The 2015 Hub-4 Sphinx-3 System
4. Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kersha, Xunying (Andrew) Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Anton Ragni, Valtcho Valtchev, Phil Woodland, Chao Zhang. The HTK Book (for HTK Version 3.5, documentation alpha version), 2015 – 72 с.
5. Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek Nagendra Goel, Mirko Hannemann, Petr Motlıcek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, Karel Vesely. The Kaldi Speech Recognition Toolkit, 2011 – 94 с.
6. Akinobu Lee, Tatsuya Kawahara. Recent Development of Open-Source Speech Recognition Engine Julius, 2009 – 57 с.
7. Adapting the default acoustic model. [Електронний ресурс] – Режим доступу: https://cmusphinx.github.io/wiki/tutorialadapt/.
8. Распознавание речи для чайников. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/226143/.
9. Распознавание речи от Яндекса. Под капотом у Yandex.SpeechKit. Електронний ресурс] – Режим доступу: https://habr.com/ru/company/yandex/blog/198556/.
10. K. -F. Lee. Context-dependent phonetic hidden Markov models for speakerindependent continuous speech recognition, 2014 – 163 с.
11. George E. Dahl, Dong Yu, Li Deng, Alex Acero. Context-Dependent PreTrained Deep Neural Networks for Large-Vocabulary Speech Recognition, 2011 – 325 с.

12. Распознавание речи. [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Распознавание_речи

13. C. -H. Lee, B. -H. Juang, F. K. Soong, L. R. Rabiner. Word recognition using whole word and subword models, 2005 – 274 с.

14. Sohn, Kim, Sung. A Statistical Model-Based Voice Activity Detection. 1999

15. T. Hughes, K. Mierle. Recurrent neural networks for voice activity detection, 2013 – 721 с.

16. Alan V. Oppenheim, Ronald W. Schafer. From Frequency to Quefrency: A History of the Cepstrum, 2004 – 348 с.

17. Shreya Narang, Ms. Divya Gupta. Speech Feature Extraction Techniques: A Review, 2015 – 292 с.

18. Jing Dong, Dongsheng Zhou, Qiang Zhang. Robust Feature Extraction Based on Teager-Entropy and Half Power Spectrum Estimation for Speech Recognition, 2015 – 183 с.

19. Data Access Object (DAO). Уровень класса. [Електронний ресурс] – Режим доступу: https://habr.com/ru/post/262243/.

20. А. А Карпов, И. С. Кипяткова, Методология оцениявания работ систем распознавания речи, 2012 – 6 с.

Додаток А

Адаптація акустичної системи до особливостей звукового сигналу

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5290_19Б

Аркушів 1

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР5290_19Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР5290_19Б	sample.package	Основні компоненти
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР5290_19Б	Додаток В.doc	Опис програмного модуля

Додаток Б

Адаптація акустичної моделі до особливостей звукового сигналу.

Текст програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР5290_19Б

Аркушів 9

Київ 2019

```
package sample;

import java.io.*;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;
import java.util.Scanner;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.text.Font;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import javax.sound.sampled.*;

public class Controller {

    private Stage stage;
    private File wavFiles;
    private File sentencesFile;
    private File transcriptionFile;
    private File dictFile;
    private File lmFile;
    private String acousticModelAbsolutelyWay;
    private String transcriptionAbsolutelyWay;
    private String sentencesAbsolutelyWay;
    private String dictionaryAbsolutlyWay;
    private String lmAbsolutelyWay;
    private String wavAbsolutelyWay;
    private int indexDir;
    private String directoryWay;
    private boolean stopCapture = false;
    private AudioInputStream audioInputStream;
    private SourceDataLine sourceDataLine;
    private CaptureThread recordThread;
    private int indexForWav =0;
    private String dict;
    private     List<ArrayList<String>>     allWords     =     new
ArrayList<ArrayList<String>>();
    private  float wcr;

//          private   ObservableList<Model>   acousticModelList   =
FXCollections.observableArrayList(listAcousticModel("C://AdaptationAco
usticModel//resource"));
    private     ObservableList<Model>     acousticModelList     =
```

```
ModelDAO.getModelList();
        private      ObservableList<Model>      sentencesList      =
SentencesDAO.getModelList();
        private     ObservableList<exampleFile>     exampleList     =
exampleFileDAO.getModelList();
        @FXML
        void openDict(ActionEvent event) {
            FileChooser fileChooser = new FileChooser();
            dictFile = fileChooser.showOpenDialog(stage);
            dictionaryAbsolutlyWay = dictFile.getAbsolutePath();
            if(dictFile != null){
                dictLabel.setText("Файл: "+ dictFile);
            }
        }

        @FXML // This method is called by the FXMLLoader when
initialization is complete
        void initialize() {
            AcousticModelComboBox.setItems(acousticModelList);
            AcousticModelComboBox.getSelectionModel().selectFirst();
            sentencesComboBox.setItems(sentencesList);
            sentencesComboBox.getSelectionModel().selectFirst();
            startRecordingBoxFirst.setItems(exampleList);
            startRecordingBoxFirst.getSelectionModel().selectFirst();
            startRecordingBoxSecond.setItems(acousticModelList);

startRecordingBoxSecond.getSelectionModel().selectFirst();
            startRecordingBoxThird.setItems(acousticModelList);
            startRecordingBoxThird.getSelectionModel().selectFirst();

            sentencesComboBox.setOnAction(new
EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    Scanner sc = null;
                    try {
                        BufferedReader  rd  =  new  BufferedReader(new
FileReader(new
File(sentencesComboBox.getSelectionModel().getSelectedItem().getCode()
)));
                        String str;
                        while((str =rd.readLine()) != null){
                            sentencesTextArea.appendText(str + "\n");
                        }
                        rd.close();
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
```

```
                }
            });

            newBut.setOnAction((new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    sentencesTextArea.clear();
                    wavTextArea.clear();
                    startRecordingTextThird.clear();
                    startRecordingTextSecond.clear();
                    exampleTextArea.clear();
                    adaptButCheckBox.setVisible(false);
                    adaptDateCheckBox.setVisible(false);
                    adaptButCheckBox.setSelected(false);
                    adaptDateCheckBox.setSelected(false);
                    indexForWav = 0;
                    indexDir                                        =
getNumDirectory("C:\\AdaptationAcousticModel\\adaptationResource");
                    directoryWay                                    =
"C:\\AdaptationAcousticModel\\adaptationResource\\Model" + indexDir++;
                    String            s            =            "cd
\"C:\\AdaptationAcousticModel\\adaptationResource\"  &&    mkdir  "  +
directoryWay;
                    enterCommandLine(s);
                }
            }));

     dictBut.setOnAction(new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    StringBuilder sb = new StringBuilder();
                    StringBuilder sb2 = new StringBuilder();
                    StringBuilder sb3 = new StringBuilder();
                    StringBuilder sb4 = new StringBuilder();
                    int i = 0;
                    for(String                retval                :
sentencesTextArea.getText().split("\n")){
                        sb.append("<s> " + retval + " </s> \n") ;
                        sb2.append("<s>   "   +   retval   +   "   </s>
(arcticWav-" + ++i + ")\n");
                        sb3.append("arcticWav-"+i+"\n");
                        sb4.append(retval + "\n");
                    }
                    try(FileWriter        writer        =        new
FileWriter(directoryWay+"\\arctic.txt", false))
                    {
                        // запись всей строки
                        writer.write(sb.toString());
                        // запись по символам
                        writer.flush();
                    }
```

```
                    catch(IOException ex){
                        System.out.println(ex.getMessage());
                    }
                    try(FileWriter        writer4        =        new
FileWriter(directoryWay+"\\arctic4.txt", false))
                    {
                        // запись всей строки
                        writer4.write(sb4.toString());
                        // запись по символам
                        writer4.flush();
                    }
                    catch(IOException ex){
                        System.out.println(ex.getMessage());
                    }
                    try(FileWriter        writer2        =        new
FileWriter(directoryWay+"\\arcticTr.transcription", false))
                    {
                        // запись всей строки
                        writer2.write(sb2.toString());
                        // запись по символам
                        writer2.flush();
                    }
                    catch(IOException ex){
                        System.out.println(ex.getMessage());
                    }
                    try(FileWriter        writer3        =        new
FileWriter(directoryWay+"\\arcticIds.fileids", false))
                    {
                        // запись всей строки
                        writer3.write(sb3.toString());
                        // запись по символам
                        writer3.flush();
                    }
                    catch(IOException ex){

                        System.out.println(ex.getMessage());
                    }
                    sb = new StringBuilder();
                    for(String                retval                :
sentencesTextArea.getText().split("\\s")){
                        sb.append(retval + " \n");
                    }
                    try(FileWriter        writer        =        new
FileWriter(directoryWay+"\\arcticDict.txt", false))
                    {
                        // запись всей строки
                        writer.write(sb.toString());
                        // запись по символам
                        writer.flush();
                    }
                    catch(IOException ex){
```

```
    System.out.println(ex.getMessage());
                    }
                    String command = "cd \"" + directoryWay + "\" &&
chcp 1251 && C:\\Users\\Alex\\Downloads\\cmuclmtk-0.7-win32\\cmuclmtk-
0.7-win32\\text2wfreq.exe          <          arctic.txt          |
C:\\Users\\Alex\\Downloads\\cmuclmtk-0.7-win32\\cmuclmtk-0.7-
win32\\wfreq2vocab.exe > arctic.tmp.vocab" +
                            " && copy arctic.tmp.vocab  arctic.vocab"
+
                            "                                                &&
C:\\Users\\Alex\\Downloads\\cmuclmtk-0.7-win32\\cmuclmtk-0.7-
win32\\text2idngram.exe -vocab arctic.vocab -idngram arctic.idngram <
arctic.txt" +
                            "                                                &&
C:\\Users\\Alex\\Downloads\\cmuclmtk-0.7-win32\\cmuclmtk-0.7-
win32\\idngram2lm.exe -vocab_type 0 -idngram arctic.idngram -vocab
arctic.vocab -arpa arctic.arpa" +
                            "                                                &&
C:\\AdaptationAcousticModel\\sphinxtrain\\bin\\Release\\Win32\\sphinx_
lm_convert -i arctic.arpa -o arctic.lm.dmp" +
                            "                  &&                perl
C:\\AdaptationAcousticModel\\ru4sphinx-
master\\text2dict\\dict2transcript.pl  arcticDict.txt  arcticDict.dict"
;
                    enterCommandLine(command);
                    adaptDateCheckBox.setVisible(true);
                    adaptDateCheckBox.setSelected(true);
                    System.out.println("Success create");
                }
            })

            adaptButton.setOnAction(new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent actionEvent) {
                    String command = "cd \"" + directoryWay + "\" &&
xcopy                                                                    "
+AcousticModelComboBox.getSelectionModel().getSelectedItem().getCode()
+            "            acousticModel            /I            &&
C:\\AdaptationAcousticModel\\sphinxtrain\\bin\\Release\\Win32\\sphinx_
fe -argfile acousticModel\\feat.params " +
                            "                    -samprate   8000   -c
arcticIds.fileids" +
                            "         -di . -do . -ei wav -eo mfc -
mswav                    yes                                             &&
C:\\AdaptationAcousticModel\\pocketsphinx\\bin\\Release\\Win32\\pocket
sphinx_mdef_convert -text acousticModel/mdef acousticModel/mdef.txt &&
C:\\AdaptationAcousticModel\\sphinxtrain\\bin\\Release\\Win32\\bw    -
hmmdir acousticModel " +
                            "        -moddeffn acousticModel\\mdef.txt
" +
```

```
                                "                                -ts2cbfn    "+
AcousticModelComboBox.getSelectionModel().getSelectedItem().getAcType(
) +
                                "                                   -feat    "+
AcousticModelComboBox.getSelectionModel().getSelectedItem().getFeat()
+
                                "       -cmn current  " +
                                "       -agc none  " +
                                "        -dictfn arcticDict.dict" +
                                "      -ctlfn arcticIds.fileids" +
                                "       -lsnfn arcticTr.transcription" +
                                "                          -accumdir   .   &&
C:\\AdaptationAcousticModel\\sphinxtrain\\bin\\Release\\Win32\\mllr_so
lve  -meanfn  acousticModel/means  -varfn  acousticModel/variances  -
outmllrfn mllr_matrix -accumdir .  ";
                        enterCommandLine(command);
                        adaptButCheckBox.setVisible(true);
                        adaptButCheckBox.setSelected(true);
                        System.out.println("Succes adapt");
                    }
                });

        startWavBut.setOnAction(new EventHandler<ActionEvent>() {
                    @Override
                    public void handle(ActionEvent event) {
                        File   outputFile   =   new   File(directoryWay   +
"\\arcticWav-" + ++indexForWav + ".wav");
                        AudioFormat                audioFormat    =    new
AudioFormat(8000.0F, 16, 1, true, false);
                        DataLine.Info                      info    =    new
DataLine.Info(TargetDataLine.class, audioFormat);
                        TargetDataLine     targetDataLine = null;
                        try {
                            targetDataLine       =       (TargetDataLine)
AudioSystem.getLine(info);
                            targetDataLine.open(audioFormat);
                        } catch (LineUnavailableException e) {
                            System.out.println("unable to get a recording
line");
                            e.printStackTrace();
                            System.exit(1);
                        }
                        AudioFileFormat.Type               targetType    =
AudioFileFormat.Type.WAVE;
                        recordThread                  =                new
CaptureThread(targetDataLine,targetType,outputFile);
                        recordThread.start();
                        System.out.println("Recording started.");
                    }
                });
```

```
            deleteWavBut.setOnAction((new  EventHandler<ActionEvent>()
{
                @Override
                public void handle(ActionEvent event) {
                    recordThread.stopRecording();
                    wavTextArea.appendText(indexForWav   +".   аудіо-
"+indexForWav+"\n");
                }
            }));

            startRecordingButFirst.setOnAction(new
EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    exampleTextArea.clear();
                    exampleTextArea.setFont(new        Font("Courier",
16.5));
                    try {
                        //BufferedReader  rd  =  new  BufferedReader(new
FileReader(new
File(startRecordingBoxFirst.getSelectionModel().getSelectedItem().getC
ode())));
                        BufferedReader  rd  =  new  BufferedReader(new
FileReader(new File(directoryWay + "\\arctic4.txt")));
                        String str;
                        int index = 0;
                        while((str =rd.readLine()) != null){
                            ArrayList<String>    arrayList    =    new
ArrayList<String>();
                            for(String retval : str.split("\\s")){
                               arrayList.add(retval);
                            }
                            allWords.add(arrayList);
                            exampleTextArea.appendText(++index  +  "  "
+ str + "\n");
                        }
                        rd.close();
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });

     private  void enterCommandLine(String command){
            ProcessBuilder  builder1  =  new  ProcessBuilder("cmd.exe"  ,
"/c", command);
            builder1.redirectErrorStream(true);
            Process p = null;
            try {
```

```
                p = builder1.start();
                BufferedReader    r    =    new    BufferedReader(new
InputStreamReader(p.getInputStream()));
                String line;
                while (true) {
                    line = r.readLine();
                    if (line == null) { break; }
                    System.out.println(line);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    startRecordingButSecond.setOnAction(new
EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    startRecordingTextSecond.clear();
                    startRecordingTextSecond.setFont(new
Font("Courier", 16.5));
                    try {
                        String command = "cd \"" + directoryWay + "\"
&&
C:\\AdaptationAcousticModel\\pocketsphinx\\bin\\Release\\Win32\\pocket
sphinx_batch -hmm acousticModel -samprate 8000 -mllr mllr_matrix -lm
C:\\AdaptationAcousticModel\\adaptationResource\\ru2.lm.dmp      -dict
arcticDict.dict -cepdir " + directoryWay + " -ctl arcticIds.fileids -
cepext .wav -adcin yes -hyp outname2.txt";
                        enterCommandLine(command);
                        String str = null;
                        File   file   =   new   File(directoryWay   +
"\\outname2.txt");
                        FileReader fr = new FileReader(file);
                        BufferedReader bf = new BufferedReader(fr);
                        int index = 0;
                        int rightWord = 0;
                        int all = 0;
                        while((str = bf.readLine()) != null){
                            for(int i = 0; i<str.length(); i++){
                                if ('(' == str.charAt(i)){
                                    str = str.substring(0, i-1);
                                }
                            }
                            List<String>       mas       =       new
ArrayList<String>();
                            for(String retval : str.split("\\s")){
                                mas.add(retval);
                            }
                            List<String> mas2 = allWords.get(index);
```

```
                                if(mas.size() >= mas2.size()){
                                    for(int i=0; i < mas2.size(); i++){

if(mas.get(i).equals(mas2.get(i))){
                                                ++rightWord;
                                            }
                                    }
                                } else {
                                    for(int i=0; i < mas.size(); i++){

if(mas.get(i).equals(mas2.get(i))){
                                                ++rightWord;
                                            }
                                    }
                                }

startRecordingTextSecond.appendText(++index + "  " + str + "\n");
                            }
                            for(ArrayList<String> m : allWords){
                                    all +=m.size();
                            }
                            wcr = rightWord/all * 100;
                            wcrSecond.setText("WCR = "+wcr);
                        } catch (FileNotFoundException e) {
                            e.printStackTrace();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
     RecFile                recFile                =                new
RecFile(startRecordingBoxFirst.getSelectionModel().getSelectedItem().g
etWav(),
startRecordingBoxSecond.getSelectionModel().getSelectedItem().getCode(
), startRecordingTextSecond, false, false);
                    }
                });

     startRecordingButThird.setOnAction(new
EventHandler<ActionEvent>() {
                    @Override
                    public void handle(ActionEvent event) {
                        startRecordingTextThird.clear();
                        startRecordingTextThird.setFont(new
Font("Courier", 16.5));
                        try {
                            String command = "cd \"" + directoryWay + "\"
&&
C:\\AdaptationAcousticModel\\pocketsphinx\\bin\\Release\\Win32\\pocket
sphinx_batch                  -hmm                    "                  +
AcousticModelComboBox.getSelectionModel().getSelectedItem().getCode()
+           "           -samprate           8000                    -lm
C:\\AdaptationAcousticModel\\adaptationResource\\ru2.lm.dmp       -dict
```

```
arcticDict.dict -cepdir " + directoryWay + " -ctl arcticIds.fileids -
cepext .wav -adcin yes -hyp outname3.txt";
                        enterCommandLine(command);
                        String str = null;
                        File   file   =   new   File(directoryWay   +
"\\outname3.txt");
                        FileReader fr = new FileReader(file);
                        BufferedReader bf = new BufferedReader(fr);
                        int index = 0;
                        int all = 0;
                        int rightWord = 0;
                        while((str = bf.readLine()) != null){
                            for(int i = 0; i<str.length(); i++){
                                if ('(' == str.charAt(i)){
                                    str = str.substring(0, i);
                                }
                            }

startRecordingTextThird.appendText(++index + "  " + str + "\n");
                        }
                    } catch (FileNotFoundException e) {
                        e.printStackTrace();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
     RecFile                recFile                =                new
RecFile(startRecordingBoxFirst.getSelectionModel().getSelectedItem().g
etWav(),
startRecordingBoxThird.getSelectionModel().getSelectedItem().getCode()
, startRecordingTextThird, false, false);
             });
         }
```

Додаток В

Адаптація акустичної системи до особливостей звукового сигналу

Опис програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР5290_19Б

Аркушів8

Київ 2019

АНОТАЦІЯ

Розділ містить опис частини, яка слугує для адаптації, що є структурною одиницею програмного продукту, та забезпечує поєднання можливостей усіх інших модулів для виконання поставлених перед системою завдань. Призначенням модуллю є обробка інформації. Модуль надає можливість отримувати необхідні дані, оброблювати та видалляти. Модуль написано мовою програмування Java, з використанням технології JavaFX.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

У додатку розглядається один з програмних модулів системи — модуль для роботи з акустичною системою з кодом УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ ТР5290_19Б, що міститься у файлі sample.package. Модуль реалізовано за допомогою програмного компонента JavaFX. Модуль призначений для управління підсистемою, яка відповідає за адаптацію акустичної моделі до особливостей вхідного сигналу. Користувач має можливість створювати нову адаптацію та перевіряти її.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Призначенням модулю для роботи з адаптацією є надання можливості створити всі необхідні адаптаційні дані, та на їх основі провести адаптацію. Використання цього модулю наддасть можливість створити набір необхідних команд, набір адаптаційних даних, аудіо файл для кожної команди та провести адаптацію.

ТЕХНІЧНІ ЗАСОБИ ЩО ВИКОРИСТОВУЮТЬСЯ

Модуль розроблено у середовищі розробки IntelliJ IDEA Community Edition 2018.3.6, що забезпечує набір архітектурних функцій та графічний діалог з користувачем, на комп'ютері, що використовував операційну систему Windows 10. Для розпізнавання мови було використано програму Pocketsphinx з використанням бібліотеки Sphinxbase. Для проведення адаптації використовувалися інструменти тренування акустичної моделі Sphinxtrain.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Програмний модуль реалізований як окремий клас, який забезпечує існування частини керування.

Для використання даного модулю не потрібно ніяких дій, оскільки він автоматично спрацьовує після запуску програмного додатку.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для модуля є інформація, яку користувач вводить в додатку. Це набір команд, запис команд в аудіо файли, акустична модель та текст для розпізнавання.

Вихідними даними програмного модуля є створена мовна та акустична модель, словник та результат адаптації.